Partial differential equations

Preconditioners based on domain decomposition appear natural for the Krylov solution of implicitly discretized ~~PDEs~~ on parallel computers. Two-scale preconditioners (involving independent subdomain solves and a global crosspoint system, as well as independent solves over interfaces of lower physical dimension) have been known since the early 1980's to be "near optimal" in the sense of providing a bounded or at most logarithmically growing iteration count as the mesh is refined. However, overall computational complexity depends on the components of the preconditioner as well as the iteration count. The cost of exact subdomain solves grows superlinearly in arithmetic complexity, and that of the crosspoint system superlinearly in communication complexity. These factors make the preconditioner granularity and the choice of its components problem- and machine-dependent compromises.

We present numerical experiments on both shared and distributed memory computers for convection-diffusion problems at modest Peclet (or Reynolds) numbers, without recirculation. Due to the development of boundary layers, these problems benefit from local mesh refinement, which is straightforward to accommodate within the domain decomposition framework in a locally uniform sense, but which introduces load balancing as a further consideration in choosing the granularity of the preconditioner. In spite of the trade-offs, cumulative speedups are obtainable out to at least medium scale granularity (up to 64 processors in our tests). Our largest problems involve about $O(10^5)$ unknowns partitioned into $O(10^3)$ subdomains and converge in tens of iterations.

(KR)

Parallel Performance of Domain-Decomposed Preconditioned Krylov Methods for PDEs with Adaptive Refinement*

William D. Gropp† and David E. Keyes‡

Research Report YALEU/DCS/RR-773
April 1990

* Also published as Argonne National Laboratory preprint MCS-P147-0490, April 1990.
† Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439. The work of this author was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, by the Office of Naval Research under Contract N00014-86-K-0310, and by the National Science Foundation under Contract number DCR 8521451.
‡ Department of Mechanical Engineering, Yale University, New Haven, CT 06520. The work of this author was supported by the National Science Foundation under Contract numbers EET-8707109 and ECS-8957475.

90 05 24 101

1. Introduction

Several tributaries of applied mathematics and computer science meet at large-scale PDE-based scientific computing applications on parallel computers, where feasibility and efficiency rank alongside existence, uniqueness, and conditioning as essential considerations.

Iterative methods based on choosing the best solution in incrementally expandable subspaces have proved effective in many contexts and have been generalized to nonsymmetric operators in such archetypal forms as conjugate gradient squared (CGS) and generalized minimal residual (GMRES). For computational economy, these methods allow use to be made of different representations of the underlying operator, ultimately converging in terms of a "high-quality" representation, through a series of applications of the inverse of a "lower quality" representation, called a preconditioner. Though already advantageous in linear problems and on serial computers, the ability to operate with multiple representations of the operator will prove even more significant in nonlinear problems and in parallel.

Adaptive discretizations determine the feasibility boundary for many scientific computing applications. Unfortunately, the irregularity of adaptivity can complicate the management of parallel resources, which has led to various compromises of adaptivity with quasi-uniformity. Generally, the use of highly structured discretizations in a modular way at different scales has been favored.

Distributed computation puts a premium on the exchange of information between cooperating processors, since such operations are generally slower than local operations. The fact that nearby points usually interact more strongly than distant points motivates decomposition by domain and allows for a "nearly" local algorithm. Nevertheless, the physics intrinsic to many PDE-based problems, expressible in the form of Green's functions with global support, requires global sharing of information. Function-space decompositions based on coarsening processes, which propagate information on multiple scales simultaneously and even asynchronously, have evolved as another active research tributary.

This contribution does not push the frontier of any particular one of these methodologies, but lies in their intersection. An implementation in software of a particular domain-decomposed local uniform mesh refinement (LUMR) generalized minimal residual (GMRES) method with a two-scale preconditioner is offered as an effective prototype, which brings various trade-offs to light. These include adaptivity and truncation error, adaptivity and convergence rate, and adaptivity and parallel load-balance. Except for details at the highest and lowest levels, the same code has been successfully ported from its serial incarnation to both shared and distributed memory computers with little sacrifice on any one machine, relative to a custom implementation for that machine alone. Our performance results on these machines will allow comparisons with other parallel PDE algorithms, not based on *a priori* domain decomposition.

The mathematical framework of our domain-decomposed linear solver is described in Section 2, followed by a discussion of the shared and distributed parallel implementations in Section 3. Section 4 contains selected performance measurements in the form of tabular cross-sections of a rather large parameter space. Finally, we draw some conclusions and comment on bottlenecks and desired extensions.

2. GMRES, LUMR, and Domain Decomposition

The generalized minimal residual method [20] is the most economical of the Krylov algorithms to which it is equivalent in the absence of round-off (including GCR [10] and ORTHODIR [23]), and is proposed for retention in [19] (along with CGNR [14] and CGS [21]) in any systematic comparative study of the performance of nonsymmetric Krylov methods. For the purpose of this study it is sufficient to recall the following salient features of GMRES, when used to solve $Ax = b$

in the right-preconditioned form $(AB^{-1})y = b$, $Bx = y$: (1) its implementation requires the ability to evaluate the action of the preconditioned operator (AB^{-1}), but not A^{-1}; (2) it converges monotonically in a number of iterations related to the number of separated clusters of eigenvalues in the spectrum (or the pseudo-spectrum, see *e.g.*, [22]) of the preconditioned operator; (3) its work and storage estimates are quadratic and linear in iteration count, respectively, because of the construction and retention of an orthonormalized basis for the Krylov subspace based on AB^{-1}; and (4) storage estimates and (sometimes) work estimates can be decreased for many problems by a restarted version of GMRES, which periodically discards the Krylov subspace.

The application of A is usually an easily parallelized operation, requiring only local communication for local discretization schemes (finite differences, finite elements, finite volumes, etc.). Good choices for B^{-1}, in the sense of producing the clustering mentioned in (2) above, will generally involve some non-local communication, but less than in parallel direct algorithms for forming the action of A^{-1}. Domain decomposition is one means of deriving efficient parallel preconditioners for elliptic operators, as well as other operator types.

Local Uniform Mesh Refinement is a means of resolving multiple spatial scales in PDE problems based on local discretizations which lies in between the extremes of excess mesh points/low overhead per point, on one hand, and minimum mesh points/high overhead per point, on the other. It is motivated by the observation that threshold levels of certain continuous function(al)s, are often good indicators for enhanced resolution requirements. Hence, adaptively inserted mesh points can be allocated in *quanta* of a convenient size and structure without a serious loss of efficiency through over-resolution. In this paper, we assume that the refinement requirements are specifiable *a priori*, but this is not an inherent limitation of the technique. Serial demonstrations of the LUMR concept have been provided by many investigators; see, *e.g.*, [1, 13] and references therein. In our implementation for a parallel port we are somewhat more restrictive than is serially natural in basing our quanta on initial coarse grid subdivisions call "tiles". Tiles have standard data interfaces with each other, but may support different discretizations and preconditioner modules internally. They are software analogs of the Geometry Defining Processors (GDPs) of [7] and lead to a code that is object-oriented in concept, but not completely so in implementation. By requiring that corners of tiles meet other tiles at corners only (and thus that tiles are elements of a logically tensor product structure) we greatly simplify the portion of the code that performs interdomain information exchange, with dividends for parallel efficiency. For standardized "docking", tiles are tensor products of half-open intervals. On average, in two dimensions, each tile "owns" one of its corners, two of its sides, and all of its interior. At physical boundaries, a tile may "own" additional pieces of its border.

The term domain decomposition spans many approaches based on a geometric partitioning of the domain of a PDE, which thus also impose a partitioning on the matrix representations of A and B^{-1}. In this contribution, we discuss non-overlapping partitionings of a two-dimensional region into subdomain interiors, subdomain interfaces, and crosspoints at interface intersections. A sample decomposition of an L-shaped region into many square subdomains is pictured in Figure 1.

We write A as

$$A = \begin{pmatrix} A_I & A_{IB} & 0 \\ A_{BI} & A_B & A_{BC} \\ 0 & A_{CB} & A_C \end{pmatrix},$$

where the submatrices arise from a simple permutation of a naturally ordered local discrete operator as follows. A_I is a block diagonal matrix representing the discretization of all of the independent subdomain interior problems, also including physical boundary points other than crosspoints. (The boundary points are retained as degrees of freedom in order to accommodate general boundary conditions, including spatial coupling and also intercomponent coupling in multicomponent problems.) In a standard five-point FD discretization of a single two-dimensional PDE, for instance, the blocks of A_I (one for each domain) are each five-diagonal matrices with the discrete subdomain diameter as the maximum bandwidth. A_B is a block diagonal matrix representing the discrete coupling *along* the artificial internal boundaries induced by the decomposition. In our five-point FD examples, each block of A_B is tridiagonal. The crosspoint matrix A_C is purely diagonal in the case of a local discretization of a single PDE. (This is based on the design assumption that the subdomains are too large to be completely spanned by a single discretization stencil.) The doubly subscripted blocks of A contain the coupling between the respective different categories of points. When neighboring subdomains are discretized at the same resolution, the appropriate coefficients are unambiguous. When resolution changes, the FE method remains unambiguous, and a version of the FV method has been likewise rendered in [18], but special provisions are required for FD discretizations. In this paper, we use a biquadratic interpolation scheme (consistent with the best accuracy to be expected from the use of a five-point FD discretization) in the subdomain interiors, when constructing a fine stencil with data requirements from a coarse region. When constructing a coarse stencil with data requirements from a fine region, we employ unweighted injection. This is a consistent discretization, but not a conservative one.

As a preconditioner, we consider

$$B = \begin{pmatrix} \tilde{A}_I & \tilde{A}_{IB} & 0 \\ 0 & \tilde{A}_B & \tilde{A}_{BC} \\ 0 & 0 & B_C \end{pmatrix},$$

where the tilde-quantities are related to the correspondingly subscripted blocks of A according to a variety of prescriptions. A key property of B is its block triangularity, which represents a compromise between a perfectly parallel block diagonality, on one hand, and the structurally symmetric form of A, itself, on the other. The formal inverse of B is, of course, also block triangular. The block B_C, solved first, is based on a coarse grid discretization of the original PDE. This step requires the exchange of data between tiles. The resulting solution provides Dirichlet endpoint conditions, through $\tilde{A}_{BC} = A_{BC}$, for the solution of the independent interface blocks which together comprise $\tilde{A}_B$. In the experiments described herein, we use for A_B a discretization of the tangential portion of the operator, those terms possessing partial derivatives along the interface in the local coordinate system. Dense but FFT-implementable interfacial blocks have also been incorporated into $\tilde{A}_B$ [4]. Finally, the blocks of $\tilde{A}_I$ are solved independently by using Dirichlet data at all artificial boundary points, through $\tilde{A}_{IB} = A_{IB}$, and actual boundary conditions at physical boundary points. In this paper, $\tilde{A}_I = A_I$, although other less expensive approximate replacements (such as fast solvers) can (and often should) be employed. Our choices for the components of B lead to good convergence results, but are not the most convenient ones for convergence proofs of Krylov iteration based on AB^{-1}. A proof of near-optimality has been given [5] for the case where the blocks of $\tilde{A}_B$ consist of the square root of the corresponding one-dimensional discrete Laplacian operator, but there is no known proof for the case employed herein, where the blocks of $\tilde{A}_B$ derive directly from the tangential terms of the overall operator.

A discussion of the utility of the block triangular form of B (as opposed to a structurally symmetric B) in the case where A is already nonsymmetric is given in [6]. In the opposite direction, the alternative of a purely block diagonal B is discussed in [12] and [17], and references therein. It may often, but not always, be dismissed as requiring too many extra iterations to justify the savings per iteration. Multicomponent problems in which source terms dominate provide examples in which a block diagonal preconditioning can outperform the coupled alternatives in overall wall-clock time.

3. Shared and Distributed Memory Implementations

The tile-based decomposition described above is motivated partly by its yield of a suitably convergent preconditioned iterative method, and partly by its control of overhead in an LUMR context, but our main interest in this contribution is its straightforward adaptation to parallel processing. There are a variety of ways to find independent threads of computation of commensurate size in the tile-based decomposition described above. Indivisible tiles can be parceled out over a MIMD processor array, or a collection of identical tiles can be broken up over a vector or SIMD processor array, to be operated on in lockstep. In between these extremes, a MIMD array can be subpartitioned into SIMD- or vector-like clusters and different sets of identical tiles allocated to each cluster for lockstep processing within each. Thus, a workload consisting of a large number of tiles in at most a moderate number of distinct sizes maps well onto a variety of parallel machines, and hybridizes well on a MIMD machine. Our implementations to date are confined to the first type: tiles are not divided over processors, but processors may be responsible for more than one tile each. The number of processors in simultaneous use is therefore bounded below by the number of tiles, but otherwise independent of it. In our current implementation, it is determined towards the outset of execution and held fixed thereafter.

In the serial implementation, work arrays for the data structures associated with each tile are allocated separately and the tiles are placed on a list which is traversed in an arbitrary given order each time a grid-oriented operation (such as forming the action of A or B^{-1}, performing a DAXPY, or taking a residual norm) is required. A highly modular approach is adopted in which each tile has its own local coordinate system and its own subroutines for problem definition, preprocessing, application of an iteration step, exchange of data with neighbors, and (as a consequence of all this flexibility) its own workload estimates for each major iterative phase. The most complex of these phases is usually B^{-1}. In the examples of this paper, the only aspect of this modularity that we take advantage of is the data exchange, in which different interface handling routines mediate the changes in mesh refinement across subdomain boundaries. However, we anticipate applying the same code to nonlinear problems such as reacting or high Reynolds number flows, in which every aspect of this modularity can purchase a significant work advantage. For instance, fast or frozen kinetics, or Euler or Navier-Stokes fluid mechanics, could be adaptively selected on each subdomain.

Code development on a serial workstation offers the obvious advantages of uncontested resources, high graphical display bandwidth, and ease of tracebacks in debugging. Furthermore, it allows the performance testing of algorithmic options, apart from communication considerations. We mention that iteration-by-iteration contour plots of solution, error (when knowable), and residual were instrumental in shortening development time.

3.1. Shared Memory Implementation

The port to a shared memory machine (the Encore Multimax 320, equipped with 18 processors) required attention to three issues beyond the serial version: data access, load balance, and work-to-processor mapping. The Multimax offers its global address space to each processor, and when the single sequential tile list traversal is broken up into several traversals of smaller tile lists, access to shared data must be controlled by locks and barriers. In anticipation of a further distributed memory port, and in keeping with a structured, minimal side-effect programming paradigm, we restricted the use of data sharing to synchronization variables, accumulation registers (for inner products or max/min reductions), work arrays for the global crosspoint system, and data buffers at the artificial interfaces of the decomposition. A list of all tiles allowing neighbor inference is also shared.

In an earlier shared memory code series for stripwise domain decompositions of tensor-product grids, see *e.g.*, [16, 17], we shared substantially more data, including entire unknown field(s). In earlier codes, data belonging to neighboring tiles was accessed by simply supplying an absolute index into the global array. In the current code, a buffer is maintained around the perimeter of each tile of a width corresponding to the semibandwidth of the difference stencil in use on that tile. These buffers are refreshed (by injection or interpolation) at appropriate synchronization points. The miserliness of the current code in allowing shared access only to interfacial buffers obviously incurs some unproductive time and space overhead, relative to the earlier versions. It is therefore natural to question whether this is an optimal shared memory code or just a distributed memory code implemented on a shared memory processor, and to assess the impact of the extra overhead on parallel performance.

To quantify the cost of this overhead, we repeatedly executed on the Multimax a loop consisting of a MATVEC (an application of A to a vector), a vector DAXPY, and a vector inner product under both globally shared and buffered versions of the stripped-down code. A five-point FD stencil with hard-coded coefficients was used for A. This is a very conservative comparison, from the point of view of establishing the validity of the buffered version as a reasonable shared memory implementation for domain decomposed elliptic PDE problems, because it includes all but one of the required forms of nonlocal data access per iterative cycle together with the minimum amount local arithmetic per access. Almost any mix of operations in a typical preconditioned iteration, except for the solution of the crosspoint system, will enjoy a better computation to "communication" ratio than these loops. Nevertheless, the penalty in wall-clock time for the buffered version was less than 10% of the globally shared version. In our judgment, this is a small penalty for the convenience of a code that is readily ported to distributed memory configurations. Of course, this is a machine-dependent conclusion. However, we note further that many current and anticipated "shared memory" multiprocessor designs have memory hierarchies which put an execution efficiency premium on global sharing. Fast global shared memory is destined to be a scarce resource on loaded multiprocessors. The Multimax 320 enjoys the evanescent luxury of a bus which is very fast relative to the data sharing requirements of 16 processors cooperating on an elliptic PDE.

Banded Gaussian elimination, used in the inversion of B_C, does not map as gracefully onto multiprocessors as do the rest of the algorithmic modules. Its parallel performance (both factorization and backsolve) has been evaluated separately from the basic loops above for a five-point stencil Dirichlet problem on the Multimax. The problem sizes varied from 16 (which corresponds to a 3×3 grid of tiles) to 4096 (a 63×63 grid of tiles), and the processing force from 1 to 16. For problem sizes up to 1024 (bandwidth 32) there is speed*down* at 16 processors, and the case of dimension 4096 is only 45% efficient at 16 processors. For our PDE examples (which employ a maximum of 1024 tiles), we therefore factor and solve the crosspoint system with single-threaded code, letting other processors idle. For a sufficiently high tile-to-processor ratio, it will become more important to parallelize the crosspoint solve itself.

Our load balancing strategy is a primitive and static one which relies on asymptotic order estimates for the largest complexity terms in both the preprocessing and iteration phases of the computation. To be specific, to an $n \times n$ tile we assign the work estimate

$$W = n^2(I^2 + I(1+n) + n^2) ,$$

where I is an estimate for the number of (non-restarted) Krylov iterations that will be necessary for convergence. The term in I^2 represents the cumulative orthogonalization work of GMRES, the two terms in I the applications of A and B^{-1}, respectively, and the remaining n^4 term the factorization preprocessing of the subdomain interior matrices of $\tilde{A}_I$.

Though I can be estimated on the basis of experience, W is sufficiently crude that we simply set it to 20 in all examples herein. There is obviously considerable room for refinement of the per-tile work estimate on the basis of *a priori* or evolving information about the tile and the overall problem. For sufficiently large tile-to-processor ratios, very balanced load distributions become possible in spite of the tile-based quantization of work, and greater effort in work estimation pays dividends.

The question of work-to-processor mapping goes beyond load balancing in the sense that the objective of load balancing is finding simultaneous executable threads of execution of comparable durations, while mapping is further concerned with the data requirements of these threads in relation to the network and memory hierarchies of the machine. From an overall performance point of view, these considerations are rarely independent, of course. We chose a fixed allocation of subdomains to processors based on either of two strategies. The first is tied to the work estimates of the previous paragraphs. Tiles are ranked in order of decreasing W and parceled out to processors without regard to spatial proximity. Once the tiles of the largest class are apportioned as evenly as possible, successively smaller classes of tiles are assigned by even tile apportionment until exhausted.

The second strategy, called "wrap mapping", is based on the heuristic argument of Section 2 which led initially to LUMR, namely, that regions requiring refinement tend to occur in contiguous clumps in elliptic PDE discretizations. Therefore, the assignment of tiles based on some regular contiguous ordering is likely to be as good as a work estimate-based mapping in the limit of large tile-to-processor ratios. The two strategies are compared in Section 4.2. Obviously, the ability to dynamically remap will be important to either mapping strategy once dynamic levels of refinement are implemented. It is clear that the work estimate strategy could be improved by allowing allocation of unequal numbers of tiles. This would lead to improvements in the results below, and should be pursued at the next level of coding sophis.ication.

3.2. Distributed Memory Implementation

The tile-based domain decomposition code has also been tested on the Intel iPSC-2SX, in a 64-processor hypercube configuration with 4 Mb of memory and a floating point accelerator (approximately 0.5 Mflops) per node. This port from shared to distributed memory was relatively straightforward because of the miserly use of shared memory in the Multimax version. The sharing of data is accomplished by explicit calls to message-passing subroutines. Since we make no assumptions about the locality in the processor domain of tiles sharing data, all inter-tile messages are routed via calls to the send/receive ethernet subroutines. For small numbers of processors, and hence strong locality of cooperating tiles, this is clearly suboptimal. However, our design is motivated primarily for extension to large numbers of processors, where going off-board is a reasonable default.

The distinctions, rooted in hardware, between "shared" and "distributed" memory are blurred by software. These should not be regarded as distinct opposites, but as different extremes on a continuum of memory hierarchies, with various cached memory designs as intermediates. At one end is the idealized model of a flat time cost for accessing different pieces of data, and at the other, a (usually nonlinear) distance- and packet-size-based time cost variation. The synchronized tile-interface buffering discussed above is well suited for the distributed memory model. In order to minimize redundant storage, data areas used exclusively in a small number of processors should not be replicated on all processors. This leads to a custom paring on each processor of the master tile data structure to the minimum size required for neighbor identification and cooperation in the global crosspoint solves and inner products. Without such storage optimizations, distributed memory processors waste significant memory.

On the iPSC it is even less inviting than on the Multimax to perform the crosspoint solve in parallel. Experiments (see, *e.g.*, [11], since updated for new hardware in unpublished form) show that a problem of size 1024 (bandwidth 32) can be speeded up slightly by parallelization on up to 32 processors, but at very small efficiency. On the other hand, the single-threaded execution used on the Multimax is also unattractive, since it would have to be preceded and followed by significant global communication to gather the right-hand side and scatter the solution. Our compromise, which dates back to our earliest hypercube domain decomposition code series [15], is to broadcast the data required for the right-hand side to each processor, then to solve the crosspoint system redundantly on each processor. A future optimization for large numbers of tiles on either shared or distributed memory computers is cooperative solution of the crosspoint system within subclusters of processors. The crosspoint problem is sufficiently different from the balance of the code in its computation to communication ratio that either extreme of employing one or all processors in it is too restrictive. Like the coarse grid solution in multigrid, it can be regarded as a black box which, though extremely important, is somewhat detachable. Many parallel algorithm designers concern themselves with this black box, and good solutions in the processor-per-point regime are readily incorporated.

4. Performance on Model Problems

We select for parallel study three two-dimensional model problems from the suite of twelve studied in serial in [13]. The examples are obtained by taking three different values of the convection, namely $c = 0$, $c = -1$, and $c = 10$, in the cylindrically-separable reentrant corner convection-diffusion problem:

$$-\nabla^2 u + \frac{c}{r}\frac{\partial u}{\partial r} = 0$$

$$u(x,y) = r^\alpha \sin\left(\frac{2}{3}(\theta - \frac{\pi}{2})\right)$$

$$\text{where } r = \sqrt{(x-1)^2 + (y-1)^2}$$

$$\text{and } \theta = \arg((x-1) + i(y-1)),\ 0 \leq \theta < 2\pi$$

$$\text{Dirichlet data on } \partial\Omega$$

$$\Omega = \text{L-shaped region}$$

The domain Ω is shown in Figure 1.

The first of these corresponds to pure diffusion, and the second and third to convection in towards the reentrant corner and away from it, respectively, at a rate inversely proportional to radius (thus satisfying mass conservation). We refer to them as the *diffusion*, the *inflow*, and the *outflow* problems, respectively. The respective values of the radial eigenfunction exponent α are $\frac{2}{3}$, $\frac{1}{3}$, and approximately 10.0442. The first two solutions of this trio lack derivatives at the reentrant corner. The last is everywhere twice-differentiable, but the solution is characterized by steep variation in three of the *non*-reentrant corner regions, where $r > 1$. Local mesh refinement is critical to improving the accuracy of a finite-difference solution. (Refinement is somewhat of a brute-force approach to a problem in which the strength of the singularity is known analytically. Instead of refinement, or in cooperation with it, a simple change to the finite-difference scheme in the vicinity of the reentrant corner can substantially improve the accuracy of the solution.)

Figure 2 displays $u(r)$ along the ray $\theta = \frac{5\pi}{4}$, which is the symmetry axis of the three L-shaped problems.

4.1. The Effect of Adaptivity on Error and Convergence Rate

The following tables provide a purely serial demonstration of the virtue of local uniform mesh refinement: comparable accuracy in considerably fewer operations, compared with global uniform refinement. We solve these problems at refinement levels of $h^{-1} = 32$, 64, 128, and 256, where h^{-1} refers to the number of equal subintervals along one of the long edges of Ω. In contrast to later tables, all of these computations were carried out to a reduction in the residual of 10^{-8}, so our report of the truncation error would not be contaminated by the error in the discrete equation iterations.

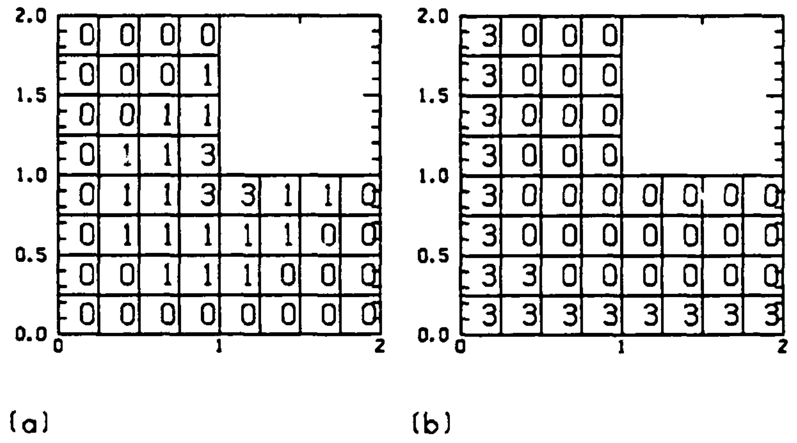

Figure 1: Sample tessellations for (a) diffusion and inflow problems, (b) outflow problem. The integer labels in each tile give the logarithm of the maximum refinement ratio employed in that tile, over all the tests. For the indicated third level of refinement, the finest tiles are 32 × 32 subintervals, while the coarsest ones are 4 × 4. Most of the test runs are performed at smaller refinement ratios. In second-level tests, all tiles showing "3" are set to "2" (16 × 16). In first-level tests, these are further reduced to "1" (8 × 8). In zeroth-level refinement, *all* tiles are set to "0".

Global refinement results are on the left, and local on the right. The finest global refinement does not appear in the table because it does not fit into the memory available, this being, of course, one of the traditional serial computing motivations for adaptive refinement. Each of the two main sets of columns lists the number of unknowns, the sup-norm of the error, the number of iterations, and the total execution time. The right-most column gives the global-to-local execution time ratios for each refinement level, where available.

The behavior of iteration count with each doubling of global refinement in the diffusion problem in Table 1 is consistent with the logarithmic growth in conditioning with h^{-1} proved for self-adjoint problems in [3]. The locally refined diffusion example inherits a sublinear growth in condition number with h^{-1} similar to the globally refined example, but the overall CPU time advantage of local refinement still increases with h^{-1}.

As expected with non-differentiable solutions (Tables 1 and 2), the sup-norm of the error shows sublinear improvement in h. The first-order accurate upwind treatment of convection leaves its signature in the outflow problem (Table 3).

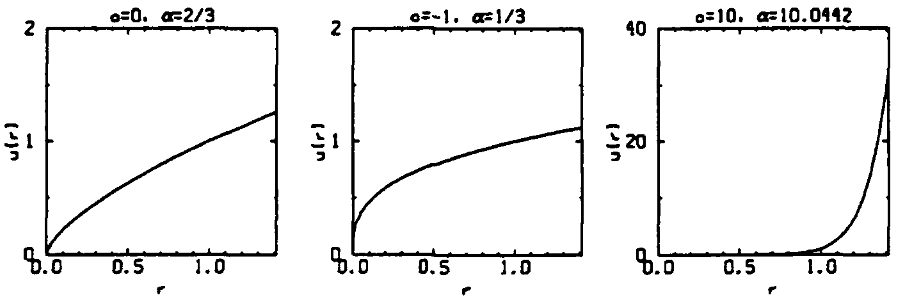

Figure 2: Cross section of $u(r)$ along the symmetry axis: (a) the diffusion problem, non-differentiable at $r = 0$, (b) the inflow problem, the singularity strengthened, (c) the outflow problem, the singularity eliminated.

	Global				Local				Ratio
h^{-1}	N_G	e_G	I_G	T_G	N_L	e_L	I_L	T_L	T_G/T_L
32	833	1.30(-2)	24	2.7	833	1.30(-2)	24	2.7	1.00
64	3201	8.30(-3)	32	6.8	1817	8.30(-3)	35	6.2	1.10
128	12545	5.25(-3)	41	27.1	2409	5.26(-3)	37	7.6	3.57
256					4745	3.33(-3)	41	16.4	NA

Table 1: Number of unknowns N, sup-norm of the error e, iteration count I, and execution time T (sec) for the diffusion problem, globally and locally refined, along with execution time ratios, for a reduction in the initial residual of 10^{-8}.

	Global				Local				Ratio
h^{-1}	N_G	e_G	I_G	T_G	N_L	e_L	I_L	T_L	T_G/T_L
32	833	6.97(-2)	23	2.6	833	6.97(-2)	23	2.6	1.00
64	3201	5.65(-2)	37	8.2	1817	5.66(-2)	34	5.7	1.44
128	12545	4.53(-2)	40	26.1	2409	4.58(-2)	37	7.6	3.43
256					4745	3.67(-2)	41	16.5	NA

Table 2: Same as Table 1, except for the inflow problem.

Case-by-case comparisons between the globally and locally refined results in Tables 1 through 3 show that the adaptive methods can be applied to representative problems with essentially no loss in accuracy, and with gains in both storage and iteration count. The gain in iteration count is modest; it is sufficient to summarize with the claim that iteration count does not noticeably deteriorate when global uniformity is abandoned in favor of local refinement to the same order, in the context of the two-level preconditioner. The gain in storage, on the other hand, is potentially very significant, and shows up both in a lower cost per iteration in already feasible problems and in an enlargement of the class of feasible problems, given a fixed-size memory. These conclusions are not new, but they are repeated here to provide self-contained motivation for the parallel results

	Global				Local				Ratio
h^{-1}	N_G	e_G	I_G	T_G	N_L	e_L	I_L	T_L	T_G/T_L
32	833	7.35(-1)	22	2.4	833	7.35(-1)	22	2.4	1.00
64	3201	4.15(-1)	28	5.7	1609	4.30(-1)	25	3.6	1.58
128	12545	2.19(-1)	34	21.5	4697	2.40(-1)	29	8.5	2.53
256					17017	1.98(-1)	35	51.6	NA

Table 3: Same as Table 1, except for the outflow problem. (The error values are large in absolute terms, but still small relative to the sup-norm of the solution. See Figure 2(c) for the scale of the solution.)

	Level 0		Level 1		Level 2		Level 3	
p	Factor	Solve	Factor	Solve	Factor	Solve	Factor	Solve
1	*1.57s*	*7.77s*	*3.33s*	*2.10s*	*6.87s*	*3.03s*	*6.04s*	*8.12s*
2	1.44	1.77	1.67	1.89	1.78	1.75	1.94	1.60
4	1.87	3.06	2.56	3.58	3.31	3.27	5.00	3.06
8	2.21	4.79	3.45	5.79	3.95	4.53	5.16	3.41
16	2.31	6.71	3.91	7.79	4.35	5.36	5.27	3.61

Table 4: Execution times (for $p = 1$) and parallel speedups (for $p > 1$) for the diffusion problem, through three levels of refinement on the Encore Multimax 320, for reduction in the initial residual of 10^{-5}. Iteration counts for levels 0 through 3 are 14, 19, 20, and 21, respectively, independent of p.

below, where the trade-offs between adaptivity and parallel load balance are addressed.

4.2. Adaptivity and Parallel Load Balance

To prevent the parameter space from growing unwieldy as multiple processors are added, we henceforth focus on the diffusion problem alone. The feature, apparent from Tables 1 through 3, that the centrally differenced diffusion problem has poorer iteration counts than the comparably sized upwind differenced convective problems makes this focus a slightly conservative one in terms of parallel performance, since the ultimate iterations of GMRES have a poorer computation to communication ratio than the early ones, because of their greater number of inner products. Overall conclusions based on studies of the inflow and outflow problems are essentially the same, however.

In the first set of tests we fix the number of tiles and vary the maximum level of refinement, h^{-1}, and the number of processors, p $(= 2^k)$. We use eight tiles along one of the long edges of Ω (as in Figure 1) and a "level-0" global resolution of $h^{-1} = 32$ (as in the first rows of Tables 1 through 3). The unrefined cases therefore employ 48 tiles, each 4×4. The uniformity makes load balancing trivial for $p = 1, 2, 4, 8$, and 16, all of which evenly divide 48. However, the small size of each tile means short threads between synchronization points. In the tables we present separately the time spent on preprocessing (predominantly factorizing the interior $\tilde{A}_I$ and crosspoint B_C blocks) and the time spent on the entire GMRES iteration (which includes the application of the preconditioner in factored form). Efficiencies (namely, the tabulated speedups divided by the number of processors) at $p = 16$ range from 14% to 46% in the preprocessing, and from 22% to 49% in the solution.

The level-0 columns are perfectly load balanced (modulo boundary effects) and hence reveal some intrinsic parallel inefficiency due to synchronization, data exchanges, and sequential data dependencies in the crosspoint equation solution. However, overall speed*down* does not occur on the Multimax or the iPSC on up to 16 processors since there is sufficient parallelizable work to offset

p	Level 0		Level 1		Level 2	
	Factor	Solve	Factor	Solve	Factor	Solve
1	*.240s*	*8.67s*	*1.16s*	*22.0s*	*3.93s*	*34.0s*
2	1.46	1.86	1.84	1.90	2.58	1.78
4	1.94	3.24	3.22	3.49	10.94	3.32
8	2.31	5.00	5.50	5.54	18.10	4.44
16	2.50	6.78	7.34	7.79	25.02	5.34

Table 5: Same as Table 4, except on the Intel iPSC/2-SX. (The most refined columns (level-3) are missing due to nodal memory limitations. The uniprocessor case does not fit.)

the inefficient crosspoint solves. It is interesting to compare the $p = 1$ rows of Tables 4 and 5 and note that the iPSC uniprocessor is faster than the Multimax in the factorization (preprocessing), but slower in the solution (iteration) stage. In spite of these uniprocessor rate disparities, both machines achieve similar speedups in the solution phase at all refinements (levels 0 through 2) on which they are directly comparable, and in the level-0 and level-1 factorization phase, as well.

The level-3 result for $p = 4$ in Table 4 shows an instance of superlinear speedup in the factorization. This repeatable phenomenon (on a devoted Multimax) is probably attributable to threshold effects in the memory caching. A far more shocking instance of superlinear speedup occurs in the level-2 factorization in Table 5. What takes 3.93s on one processor takes 0.157s on 16 processors, for 156% efficiency. The observation is also repeatable, and is probably due either to a caching effect in the feeding of the Weitek co-processor or to collisions among auto-referential messages in the cases of small numbers of processors (see the beginning of Section 3.2).

The generally poor efficiencies for $p > 4$ in the level-2 and level-3 problems in Table 4 result from the fact that there are only three of the largest size tiles to go around (see Figure 1(a)). Hence, all but three processors are idle for significant periods of the computation. Because of the small tile-to-processor ratio in this problem, the load becomes more and more unbalanced at high refinements and large numbers of processors. This phenomenon shrinks the "paydirt" region of parameter space for local uniform refinement in the parallel setting. For a problem in the small tile-to-processor ratio regime which would otherwise lead to a lumpy work distribution, a certain amount of extra (numerically unnecessary) refinement, padding out the work distribution, is "cheap". The extra storage it requires is presumed available on a homogeneous distributed memory array, the extra data movement is likely to be masked by comparably sized messages, and the crosspoint system is unaffected by the degree of refinement.

The parallel efficiencies above are not impressive, but consideration should be given to the fact that the usual motivation for large-scale algorithm design is rapid solution, not linear speedup. To illustrate the potential incompatibility between these objectives, consider Table 6, which resolves the same diffusion problem at a finer resolution ($h^{-1} = 128$) for a variety of tile sizes and processor numbers. These problems are too large for examination over the full range of p, because along with the hardware advantage of evenly distributed memory comes the disadvantage (relative to a shared memory machine) that subsets of the total number of processors have less aggregate memory. The relative speedup tabulated in these examples is the reciprocal of the ratio of each runtime to that in the previous row.

The best speedups are in the data set occupying the first three rows, which has the sparsest crosspoint system and the largest tiles. However, this set employs the smallest number of processors and on average yields the worst wall-clock times. The next set has enough medium-sized tiles to employ up to the maximum number of processors available, and it gives the best wall-clock times, in spite of mediocre efficiency. The last set uses tiles that are too small, and it is choked by the

Tessellation	p	Factor Time (s)	Factor Rel. Sp.	Solve Time (s)	Solve Rel. Sp.	Total Time (s)	Total Rel. Sp.
48 tiles	4	11.61		54.91		66.52	
each	8	5.94	1.95	28.61	1.92	34.55	1.93
16 x 16	16	3.13	1.90	15.30	1.87	18.43	1.87
192 tiles	8	2.28		16.95		19.23	
each	16	1.71	1.33	9.91	1.71	11.62	1.65
8 x 8	32	1.42	1.20	6.37	1.56	7.79	1.49
	64	1.28	1.11	4.67	1.36	5.95	1.31
768 tiles	16	15.79		19.92		35.71	
each	32	15.73	1.00	14.62	1.36	30.35	1.18
4 x 4	64	15.70	1.00	11.83	1.24	27.53	1.10

Table 6: Execution times and relative parallel speedups for the diffusion problem, globally refined to $h^{-1} = 128$ on the Intel iPSC/2-SX, for reduction in the initial residual of 10^{-5}. Iteration counts for the successively finer tessellations are 24, 16, and 11, respectively, independent of p.

Tessellation	p	Factor Time (s)	Factor Rel. Sp.	Solve Time (s)	Solve Rel. Sp.	Total Time (s)	Total Rel. Sp.
48 tiles	2	10.31		64.85		75.16	
each	4	5.23	1.97	33.39	1.94	38.62	1.95
8 x 8	8	2.35	2.23	19.50	1.71	21.85	1.77
	16	1.29	1.82	12.63	1.54	13.92	1.57
192 tiles	8	1.60		13.33		14.93	
each	16	1.31	1.22	8.38	1.59	9.69	1.54
4 x 4	32	1.22	1.07	6.00	1.40	7.22	1.34
	64	1.16	1.05	4.85	1.24	6.01	1.20

Table 7: Same as Table 6 except locally refined (level-1) to $h^{-1} = 128$. Iteration counts for the successively finer tessellations are 27 and 18, respectively, independent of p.

large crosspoint system. In terms of total execution time, this tessellation is worse at 64 processors than is the first at 16 processors. In [13], it was shown for a variety of convection-diffusion problems that the optimum granularity of tessellation (for a two-scale preconditioner with exact subdomain solves) occurs between the extremes of many small tiles (dominated by the crosspoint system) and few large tiles (dominated by the interior systems).

Table 6 is for uniform global refinement. The same effective resolution is achievable at level-1 refinement with a coarse grid, as recorded in Table 7. Again the 192-tile partitioning is most successful in terms of wall-clock time, although the parallel efficiencies for this case are again mediocre. It is interesting to observe that the best time for this problem, 6.01s for the 64-processor run, is within 1% of the best time on the globally refined problem of the same effective resolution; in fact, it is 1% worse. The serial time for the locally refined case would of course be less, but load imbalance resulting from a small tile-to-processor ratio (of 3) in this case takes away the advantage of local refinement in parallel. We do not expect major synergistic advantages in the combination of local refinement and parallel processing until we encounter larger problems and/or write better work estimate and work mapping algorithms.

Tessellation	N	p	Work Estimate Factor (s)	Work Estimate Solve (s)	Wrap Factor (s)	Wrap Solve (s)
768 tiles ea. 4 × 4	25,969	16	16.52	34.40	16.24	35.45
		32	16.07	23.34	15.95	24.16
		64	15.85	18.09	15.78	18.49
768 tiles ea. 8 × 8	103,201	32	23.70	69.48		
		64	19.71	45.23	17.99	48.55

Table 8: Execution times for the factorization and solution phases of the tile algorithm based on two different tile-to-processor mappings. Iteration counts for the successively finer refinements are 15 and 20, respectively, independent of p.

Finally, we investigate differences between the work estimate and the wrap mappings of tiles to processors. This study is presented for a level-1 refinement at the upper end the problem sizes we have tested in two dimensions. In Table 8 we consider some p = 16, 32, and 64 cases on the iPSC for an effective resolutions of $h^{-1} = 256$ and 512 and varying numbers of tiles per side. We observe that the wrap mapping is slightly better for balancing factorization work and slightly worse for the balancing of the iteration work. On the whole, however, it is interesting that as simple a scheme as wrap mapping competes as well as it does with a first attempt at a work estimation scheme.

It is also interesting to study further breakdowns of the factorization and solution times. Of the 19.71s required in preprocessing, the largest problem under the work-estimate-based mapping, 79% was devoted to factoring the crosspoint system, 20% to the subdomain interior systems, and 1% to the interface systems. Of the 45.23s required to iterate it to convergence, 35% was spent solving the crosspoint system, 22% solving the interior systems, and 1% solving the interface systems. The remaining solution time went into the work of the GMRES algorithm apart from the inner products (23%), the inner products themselves (13%), and the application of A to a vector (6%).

We also note that the globally refined problem which is equivalent to the last row of the table requires storage of 197,633 unknowns (nearly double the locally refined total of 103,201), and requires 84.00s in total execution time (as opposed to the 64.94s of the work estimated mapping run). Thus, as we begin to approach large problems, LUMR begins to pay dividends even in parallel.

5. Summary and Shopping List

We conclude with some brief assessments of the state of parallel domain decomposition algorithms.

The potential advantages of domain-decomposed preconditioned iterative methods for PDE problems with complex geometry, adaptive refinement requirements, and vast numbers of unknowns are obvious. Our experiments demonstrate that a two-level preconditioner which includes a global crosspoint solve is capable of maintaining good conditioning, thus providing good algebraic convergence and preserving the usefulness of double-precision computer arithmetic in the context of very large problems. The key features to be addressed in a parallel implementation are the inefficiency of the crosspoint solve itself, and load imbalances.

The communication-intensive crosspoint system is the single largest consumer of time in both the preprocessing and iteration phases, when the number of tiles is sufficiently large; and a large number of tiles is often required to realize the advantages of the method in complex geometry and adaptive refinement contexts. Thus, the distributed solution of sparse matrix problems should remain the focal point of research that it is today. It is the importance of having a conveniently structured crosspoint system that keeps us from a much looser framework for our tessellations. This is described in greater detail in [13]. A preconditioner based only on nearest neighbor interactions would impose almost no rules on the topology of the decomposition, but the deterioration in convergence rate in the many tile limit would eventually become unacceptable, and no further useful parallelism could be found without subdividing the tile.

Multigrid solution of the crosspoint system might be a useful alternative to direct solves with large numbers of tiles. However, multigrid itself requires a "sufficiently fine" coarse grid in the presence of skewsymmetry in the operator A, so the problem is rather universal. Furthermore, the problem becomes worse in three dimensions. Available theory for the case of non-overlapping subdomains suggests that the crosspoint solve must be generalized to a "wire basket" solve (including implicitly the edges between the vertices) in order to maintain near optimal conditioning in three dimensions; for a discussion, see [9]. An alternative with better parallel computational complexity is given as "Method 1" of [2]. The preconditioner in this case consists of a global solve for a mean value on each substructure, along with independent edge, face, and subdomain solves. The matrix operator for the mean value problem has the same size and connectivity as a standard three-dimensional finite difference problem on a coarse grid but it is not derivable in as simple a manner as the crosspoint system employed herein, as a rediscretization of the original operator. Additive Schwarz-type preconditioners (see [8]) with overlapping subdomains avoid interface problems altogether, and may prove more convenient for general three-dimensional problems. They do not avoid the global crosspoint system, however, and since the interface problems are subdominant terms in the computational complexity, parallel experience with the tile algorithm is likely to provide good design rules for additive Schwarz codes, as well.

Load imbalance can seriously detract from the parallel performance of an adaptive algorithm. It is desirable to have mapping algorithms that are superior to the simple ones employed herein, but they obviously must not be so complex as to begin to dominate the total execution time, themselves. Attention to the processor locality of frequently cooperating (*i.e.*, neighboring) tiles could also yield important dividends, depending on the architecture.

Tile subdivision and the clustering of homogeneous tiles in order to form long vectors whose components can be processed in a SIMD mode are two modes of parallelism that we have yet to investigate, although their promise is considerable. Neither would necessarily replace the indivisible heterogeneous tile algorithm evaluated herein; rather, they could be hybridized with it to move into parallel realms beyond moderate granularity MIMD. Tile subdivision accommodates numbers of processors larger than the maximum useful size of the crosspoint system. Homogeneous tile clustering accommodates vector nodes in a MIMD array or vector-like SIMD clusters within a MIMD array, the latter hybrid architectures being likely for massively parallel supercomputers because of their cost-effective use of the silicon.

References

[1] M. J. Berger & J. Oliger, *Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations*, J. Comp. Phys., 53 (1984), pp. 484–512.

[2] J. H. Bramble, J. E. Pasciak & A. H. Schatz, *The Construction of Preconditioners for Elliptic Problems by Substructuring, IV*, Math. Comp., 53 (1989), pp. 1–24.

[3] ————, *The Construction of Preconditioners for Elliptic Problems by Substructuring, I*, Math. Comp., 47 (1986), pp. 103–134.

[4] X.-C. Cai, *Personal communication*, 1990.

[5] X.-C. Cai, W. D. Gropp & D. E. Keyes, *Convergence Rate Estimate for a Domain Decomposition Method*, 1990. (Manuscript).

[6] T. F. Chan & D. E. Keyes, Interface Preconditionings for Domain-Decomposed Convection-Diffusion Operators, R. Glowinski, et al., eds., *Third International Symposium on Domain Decomposition Methods*, SIAM, Philadelphia, 1990.

[7] D. Dewey & A. T. Patera, Geometry-Defining Processors for Partial Differential Equations, B. J. Alder, ed., *Architectures and Performance of Specialized Computer Systems*, Academic Press, New York, 1988.

[8] M. Dryja & O. B. Widlund, *An Additive Variant of the Schwarz Alternating Method for the Case of Many Subregions*, Technical Report TR 339, NYU, Courant Institute, December 1987.

[9] ————, *Some Domain Decomposition Algorithms for Elliptic Problems*, Technical Report TR 438, NYU, Courant Institute, April 1989.

[10] H. C. Elman, *Iterative Methods for Large, Sparse, Nonsymmetric Systems of Linear Equations*, Technical Report RR-229, Yale University, Dept. of Comp. Sci., April 1982.

[11] W. D. Gropp, *Solving PDEs on Loosely-Coupled Parallel Processors*, Par. Comput., 5 (1987), pp. 165–173.

[12] W. D. Gropp & D. E. Keyes, *Domain Decomposition on Parallel Computers*, Impact Comput. Sci. Eng., 1 (1989), pp. 421–439.

[13] ————, *Domain Decomposition with Local Mesh Refinement*, Technical Report RR-726, Yale University, Dept. of Comp. Sci., August 1989. Submitted to SIAM J. Sci. Stat. Comp.

[14] M. R. Hestenes & E. Stiefel, *Methods of Conjugate Gradients for Solving Linear Systems*, J. Res. Nat. Bur. Stand., 49 (1952), pp. 33–53.

[15] D. E. Keyes & W. D. Gropp, *A Comparison of Domain Decomposition Techniques for Elliptic Partial Differential Equations and their Parallel Implementation*, SIAM J. Sci. Stat. Comp., 8 (1987), pp. s166-s202.

[16] ————, Domain Decomposition Techniques for Nonsymmetric Systems of Elliptic Boundary Value Problems: Examples from CFD, T. F. Chan, R. Glowinski, J. Periaux & O. Widlund, eds., *Second International Symposium on Domain Decomposition Methods*, SIAM, Philadelphia, 1989.

[17] ————, *Domain Decomposition Techniques for the Parallel Solution of Nonsymmetric Systems of Elliptic BVPs*, 1990. Appl. Num. Meths. (To appear).

[18] S. F. McCormick, ed., *Multilevel Adaptive Methods for Partial Differential Equations*, SIAM, Philadelphia, 1989.

[19] N. M. Nachtigal, S. C. Reddy & L. N. Trefethen, *How Fast are Nonsymmetric Matrix Iterations?*, Technical Report Numerical Analysis Report 90-2, Massachusetts Institute of Technology, Dept. of Mathematics, March 1990. Submitted to SIAM J. Sci. Stat. Comp.

[20] Y. Saad & M. H. Schultz, *GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems,* SIAM J. Sci. Stat. Comp., 7 (1986), pp. 856–869.

[21] P. Sonneveld, *CGS, A Fast Lanczos-Type Solver for Nonsymmetric Linear Systems,* SIAM J. Sci. Stat. Comp., 10 (1989), pp. 36–52.

[22] L. N. Trefethen, Approximation Theory and Numerical Linear Algebra, J. C. Mason & M. G. Cox, eds., *Algorithms for Approximation,* Chapman, 1990.

[23] D. M. Young & K. C. Jea, *Generalized Conjugate Gradient Acceleration of Nonsymmetrizable Iterative Methods,* Lin. Alg. Appl., 34 (1980), pp. 159–194.